Models for Replica Synchronisation and Consistency in a Data Grid

Dirk Düllmann, Wolfgang Hoschek, Javier Jaen-Martinez, Ben Segal
CERN, European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland
{Dirk.Duellmann, Wolfgang.Hoschek, Javier.Jaen-Martinez, Ben.Segal}@cern.ch

Asad Samar
California Institute of Technology, Pasadena, CA 91125, USA
Asad.Samar@cern.ch

Heinz Stockinger, Kurt Stockinger
CERN, European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland
Inst. for Computer Science and Business Informatics, University of Vienna, A-1010 Vienna, Austria
{Heinz.Stockinger, Kurt.Stockinger}@cern.ch

Abstract

Data Grids are currently proposed solutions to large scale data management problems including efficient file transfer and replication. Large amounts of data and the world-wide distribution of data stores contribute to the complexity of the data management challenge. Recent architecture proposals and prototypes deal with replication of read-only files but do not address the replica synchronisation problem. We propose a new Grid service, called Grid Consistency Service (GCS), that sits on top of existing Data Grid services and allows for replica update synchronisation and consistency maintenance. We give models for different levels of consistency provided to the Grid user and discuss how they can be included into a replica consistency service for a Data Grid.

1 Introduction

Recently, Data Grids [2] have become an interesting and popular domain in the Grid community. In particular, the management of huge amounts of data is one of the major scientific challenges to be addressed by [14]. A typical Data Grid can have Terabytes or even Petabytes of data distributed and replicated all around the globe. In this paper, we concentrate on models for high-level replication services, namely services for maintaining replica synchronisation and consistency. Such services can be built on top of existing replication services for fast file transfer (e.g. GridFTP [9] and file meta-data management (i.e. replica catalogues [2])). This is in-line with Grid architectural considerations [8] where replica consistency issues concern high-level services that need to be adapted to the application environment. Lower level services, on the other hand, guarantee efficient storage, retrieval of replica location information, and efficient file transfer.

Since replication can be interpreted in many different ways, we define that high-level replica management deals with consistency aspects of replicas. A replica is not just a simple copy of a file. For instance, one might copy a file into local, temporary disk space without making the file available to the Grid. This is what we call a simple copy. As regards a replica, two or more physical file instances of the same logical file have to be synchronised and some meta information is kept that knows about both replica locations. Update operations have to be consistently propagated to other replicas. We claim that an efficient replication mechanism is driven by two factors: knowledge about the data and use cases - both are specific to the application domain. We guide our replication discussion by High Energy Physics requirements and propose replica consistency models that are applicable to several Data Grids.

In general, the data consistency problem deals with keeping two or more data items, in our case replicas, up to date, i.e. consistent. A strict approach guarantees that all replicas are always 100 percent in sync and thus fully consistent. Due to the locking overhead of keeping huge amounts of distributed data in sync [11], 100% consistency is a very impractical solution for a Grid environment. Thus, if knowledge about the data and user requirements (use cases) are available, one can relax this strict consistency requirement and allow certain parts of the data to be out of sync for a particular amount of time. For instance, a site A in a Data

Grid says explicitly that newly created files at other sites B, C, and D have to be transferred to the site A within two days. This means the replica creation process can be done within a 48 hour time frame. Within this period the state of physical files can be inconsistent. Another example is that writable replicas have to be updated and synchronised every 10 minutes. As a third example, we mention that updates of meta information might be 100 percent synchronised. This clear need for different consistency models is the key input to our paper and guides the proposed solutions.

The paper is organised as follows. Section 2 discusses the current state-of-the-art Grid architecture for replication issues as it is also used in recent projects. The High Energy Physics data model and use cases are described in the next section in order to put the replication effort into the right context. Our proposed Grid architecture for replica synchronisation is given in Section 4. It is mainly based on dynamic elements in a Data Grid. Section 5 discusses some general replication issues like the level of replication and a/synchronous replication mechanisms. Our main models are discussed in terms of consistency levels in Section 6 and merged with replication protocols in the following section. We finally conclude our paper and give some future work.

2 Current Data Grid Replication Architecture

Distributed systems and distributed database management systems provide several generic replication models that mostly work well for local-area replicated data. Real-time applications with high requirements for data consistency make use of these models [15]. In emerging Data Grid applications, we can identify some fundamentally different features and application requirements in contrast to real-time database applications like banking or finance applications. For a more detailed discussion on the state-of-the-art in distributed database replication refer to [23].

- Data is distributed and replicated to world-wide distributed sites: wide-area replication rather than local-area replication (in relatively small environments). This has an implication on data consistency since wide-area networks in general have longer latencies.

- Most of the data is read-only. If updates occur, general real-time update propagation mechanisms are not needed (or needed only in relatively small sub-environments).

- A general replication system is required that sits on top of a database management system or data store since a broad user community will use several different storage technologies. The approach should be applicable to several systems but not necessarily combine them.

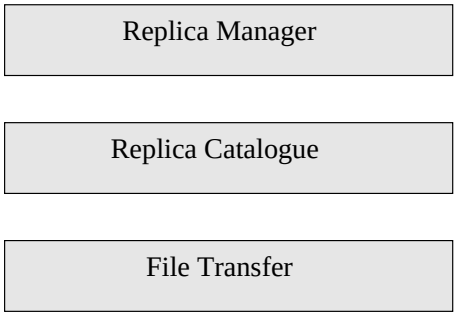

Figure 1. Architecture of the Replication Services in a Data Grid

Thus, a Data Grid cannot rely on a replication strategy provided by a single database management system like Oracle [17] or Objectivity [16] but needs to build synchronisation models and tools that are applicable to a large variety of heterogeneous data stores. Since Grid technology is middleware technology, a replica update synchronisation system has to be provided as middleware between the end-user application and the actual data stores. In principle, there are possible candidates in commercial systems like Oracle or Objectivity but they are either not specialised for wide-area replication or only for a homogeneous database management system.

A general architecture for a Data Grid is given in [8]. The building blocks and services of the replication sub-system are illustrated in Figure 1.

A clear separation of services allows for a flexible system. Currently, data replication in Grids mostly deals with file replication. We start off with this model and enhance it by our proposed replica synchronisation service.

The lowest layer of the architecture above deals with efficient data transfer between two sites. For file transfers, several protocols exist: FTP [18] and HTTP [7] are the most common file transfer protocols. In the Grid community, GridFTP [3] - an enhanced version of FTP - is a viable candidate for a new standard. Consequently, in our replication model we assume that the file transfer problem is sufficiently dealt with and we build on top of it.

Replica catalogues are responsible for locating physical files and mapping of logical filenames to physical instances. The replica manager layer is responsible for efficiently creating, moving and deleting replicas in the Grid. In detail, files are copied from one site to another and registered in the replica catalogue. This makes files available to the Grid.

This architecture works well for static environments where files are read-only, i.e. the architecture satisfies only

a particular use case: read-only data. Prototypes that implement parts of the architecture have been made [21] and show good results, i.e. the software satisfies the needs in a production environment.

We now go a step further and introduce more dynamic elements into the architecture: files and meta-data can be updated. There is a clear need for a Grid service that takes care of these requirements and provides efficient solutions. The main contribution of this paper is to provide first models for replica synchronisation in a Data Grid. Before a more detailed discussion on the proposed service, we analyse the dynamic elements in the Data Grid. We claim that efficient replication can only be reached if the environment, the data model and possible use cases are well understood. In the next section we discuss the High Energy Physics data model and use cases as an example for a Data Grid.

3 The High Energy Physics Data Model and Use Cases

Current Data Grids like DataGrid [6], GriPhyN [12] and Particle Physics Data Grid (PPDG) [19] support several scientific application domains. They all have in common the High Energy Physics (HEP) community that we want to use as an example for our replication models.

3.1 Data Model

The HEP data model foresees different types of data with essentially different characteristics and requirements as regards data access and data management. Briefly, **raw data** is the original data that is produced by a particle physics detector. This data is further processed and so called **reconstructed data** are produced. The reconstructed data can further be divided into **Event Summary Data (ESD)** and **Analysis Object Data (AOD)**. The smallest data type we can distinguish is the so called **tag data** that stores summary information about raw and reconstructed data objects. In general, all these different types of data vary in size and are read-only. Additional data types are event meta-data (read/write) and calibration data (read/write).

3.2 Use Cases

The following use cases influence the way data is accessed, distributed and replicated.

3.2.1 End User Physics Analysis

End user physics analysis will be one of the main jobs of a physicist trying to understand the physics properties of the data that is produced by the particle detector. Even though this kind of analysis will be mainly based on data like tag, AOD and ESD, it will also require various types of meta data like detector calibration data, geometry data and indices to support fast data access. Output data will have different meta data and may be integrated with the "main store", i.e. files have to be registered in a replica catalogue.

This use case can be regarded as **User Driven Replication**: in other words, the user has the knowledge of what his analysis job is doing and should decide how and what to replicate. Besides, he might not want any type of synchronisation between files and the master copy. (Note that the master copy is the original file created by a data production process.) An automatic replication procedure is not possible here since not all information is available in advance for automatically triggering a replication process.

Although most of the data that the end-user deals with is read-only, the meta-information related to this bulk data is modifiable. The changes in this meta-data have to be propagated to the different replicas and so we need at least some level of consistency between them.

3.2.2 Distributed Simulation

Another typical use case is distributed simulation. Several sites in the Data Grid generate simulated data according to some physics algorithms. This so-called Monte Carlo simulation is characterised by small amounts of input data (mainly parameters, geometry etc.) and larger amounts of output data (also read-only). The output needs to be integrated into the main store of replicas.

We refer to this simulation process as **Data Shipping:** A production centre produces read-only data (single-writer data). This data is replicated (shipped) to other sites that in turn produce and replicate (ship) data. We can outline the following different requirements and thus policies (see Section 5.1) for replication:

- Site A needs full synchronisation of its replicas with all the data produced at site B.
- Site C needs full synchronisation of its replicas with only a subset of the data that is produced at site D.
- Sites E, F, G request asynchronous replication (replication on demand) of site H.

4 A Replica Consistency Service

In this section the proposed replica consistency service with respect to the update problem is discussed.

4.1 Dynamic Elements and Consistency

An important point for update synchronisation models is to understand the nature of updates, which data is affected

and how this influences the end-user application. Based on the general Grid architecture in Section 2 and the discussion in the previous section, we can identify the following dynamic factors in a Data Grid:

- File and replica catalogues are updated whenever new files are introduced, deleted or moved. If a replica catalogue is a single, central service using a conventional database management system (DBMS) for storing logical and physical file information, the DBMS can take care of the update problem. If the replica catalogue is distributed or replicated itself, the individual catalogues need to be synchronised. One example: every time a new logical file is introduced, the filename needs to be unique within all replicas of the catalogue - a clear case for synchronisation. We regard this as meta-data synchronisation since the replica catalogue is meta-information on the actual data. In the scope of this paper, we assume that this meta-data replication is taken care of by the replica catalogue service and concentrate on the synchronisation of the actual data. However, the problem to be addressed is almost the same. For instance, the replica catalogue system can use the consistency service to keep replica catalogues consistent.

- Update of file contents. This is the classical replica update problem. Replicated data is updated at one place and the changes are propagated to all other replicas. In detail, a file is opened for write access, bytes are added or changed and closed afterwards. Within a certain amount of time these changes have to be visible at all replicas in order to have a consistent view of the data. Possible solutions are given in the next section.

4.2 The Consistency Service in Detail

Replica update propagation is done in order to guarantee a consistent view of replicated data. We define a service that provides update synchronisation as a *consistency service* and regard it as an additional Grid service. An application that wants to update data and propagate these updates to other sites in the Data Grid needs to use this service. Sites that have subscribed to the service will be included in the update synchronisation process.

A consistency service (see Figure 2) needs to take into account different replication schemes and consistency models ranging from fully synchronised data to loosely synchronised sites where updates and changes are visible only after a few minutes, hours or even days. Thus, the end-user can choose a consistency level that satisfies his application requirements. Possible consistency levels are discussed in Section 6.

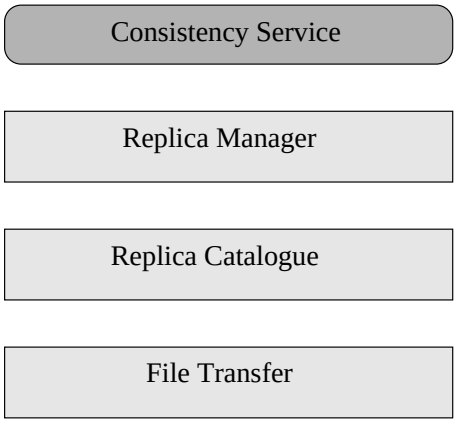

Figure 2. A Replication Architecture including the consistency service.

The consistency service shall provide general consistency levels and interfaces but the actual service implementation is tightly coupled with the underlying data store and data model. Whereas in a relational DBMS a standard query interface like SQL can be used to handle updates at a local site and propagate them efficiently to all replicas, a data store like Objectivity or Root does not provide such a high-level interface and thus requires more sophisticated update mechanisms. Some proposals can be found in [23].

To sum up, when a database management system is used locally, the DBMS has to guarantee local consistency whereas the replication middleware has to take care of global consistency in the Data Grid. Thus, global consistency deals with propagation of update information to remote sites in the Grid.

5 General Replication Issues

5.1 Replication Protocols, Policies and Simplifying Boundary Conditions

In principle, two mainly different replication approaches are known: synchronous and asynchronous replication. Whereas synchronous replication aims for keeping all the replicas permanently in sync, asynchronous replication allows for a certain delay in updating replicas. Based on the relative slow performance of write operations in a synchronously replicated environment (due to the strict two-phase commit protocol that is used within a database transaction [11]), the database research community is searching for efficient protocols for asynchronous replication accepting lower consistency. For a detailed discussion on a/synchronous replication refer to [23].

Several replication use cases are possible and the amount of read and write access to data influences the replication policy. It is very likely that various boundary conditions will affect the replication and allow for simplifications.

read-only data: The simplest case is if data is read-only, where data may be copied (1) at any point in time (2) from any replica to any other place. This requires no locking nor any other coupling (except for the replica catalogue) of replicas.

Note it is probably very hard to ever remove the read-only property from a file in a running system without risking to compromise readers. Therefore, applications would be required to insure that data will never need any change.

writable data: Once we allow write access to the data, it is important to have a clear policy that defines who is allowed to write/change data. If ownership is assigned to files (replicas), one policy can be that only the owner is allowed to modify the original version of a file (master copy). For a data item which can be updated (writable) we distinguish between permanent and varying ownership.

- **well defined file ownership ("master-slave case"):** Only one well defined entity in the system (e.g. one user, or a production team at one site) is allowed to modify a particular piece of data (e.g. a file). As a result, the replication is not symmetric any more between all replicas in the system. The process of determining which is the most up-to-date version in the system is not required. Only the information "who is the owner" needs to be propagated to all slave replicas. In case of data access only one well defined node needs to be contacted to obtain the most recent version of the data.

  This is only true for write operations. For a read access, any replica can be selected since the master-slave approach guarantees that all copies are up-to-date [23]. In detail, all write and update requests are forwarded to the master which in turn is responsible for synchronising all the slaves. Read requests can be served by any replica [23].

- **varying writers (no central control of replicas):** This is the most general and complex case. Several update operations need global agreement between all replicas and will also try to contact all replicas to obtain a quorum. Quorum systems are commonly used as a mechanism to get the right, for example, to update a replica. The current distributed database research proposes several solutions to this problem. For a more detailed discussion refer to [23].

5.2 The Level of Replication

There exist two different replication policies as regards the amount of data to be replicated: *full* and *partial replication*. A fully replicated system replicates all data items to all participating sites. In a partially replicated environment only a subset of all data items is replicated to some or all sites. Furthermore, we can distinguish between the replication of:

- a complete site (we assume that a site is holding a finite number of data items or files)
- a consistent subset of files of a site (files that are logically connected)
- individual files (files that are independent of each other)
- collections of individual objects in files (these objects may or may not be consistent subsets)
- other exchange file formats

The following question can be raised: "Can we copy just single, independent files or do we need to take into account that they are interrelated?". In principle, it is the individual application that decides what data has to be replicated. However, replicating individual sets of objects in a file can cause, for instance, logical connections between the objects to get lost (broken links, dangling pointers). A good replication mechanism guarantees that the application user should not need to care about what data is needed and whether it is in a safe state or not.

We would not always like to deliver the full set of reachable objects to the users. Depending on their object model this could be impossible anyway since this approach might result in very large volume transfers. In the worst case, a complicated object and association structure has relations to all files and thus all files need to be replicated in order to navigate through all possible associations.

6 Data Consistency Levels Delivered to Grid Users

In the following section we describe several possible consistency levels and discuss their usefulness. Database theory [10] provides valuable background and solutions to some of the problems. Our discussion here is guided by database transaction theory including locking for establishing consistent data. Higher consistency levels including transactional integrity across a multi-user, multi-file store typically require using a database system like [16] or [17].

We therefore also discuss how replication could be integrated with database management systems to retain some or all of their consistency guarantees.

For non-database stores, which may not provide transactional consistency or may not support concurrent read/write access (e.g. ROOT [25]) , some of the following consistency levels may not be applicable. Their replication model is often based on the simplifying condition (see Section 5.1) of assuming completely read-only data which is sufficient for many applications.

It should also be pointed out that several of the problems and solutions which are discussed here have already been discussed in the context of the RD45 [20] project at CERN. The replica consistency problem is very similar to the consistency problem for (partial) database backups [5].

6.1 Possibly Inconsistent Copy (Consistency Level -1)

The file replica is created using a trivial file copy concurrently with ongoing write operations. For illustration let us assume a file is located at a particular site and a database management system or a file system operation takes place on that file. While one user is updating the existing file, another user is copying the file to another location. This corresponds to the classical "dirty read" problem where a reader is accessing a file while another one is writing to the same file. A file copy corresponds to a read operation since each copy instruction starts with reading the file context before sending it via a socket connection to another location (as is the case for FTP).

The resulting file does not necessarily correspond to a state of the original file at *any* point in time and internal data structures may be inconsistent.

Clearly, this is of limited use to Grid end users. One could neither guarantee that any user job does not suddenly fail, nor could one exclude that it finishes by delivering incorrect analysis results.

There are several well known ways to tackle this problem:

standard locking: obtain a file write lock - perform the file copy - release the lock.

optimistic locking: In case of a very low probability of lock contention on the file, one could alternatively copy without getting a lock and test the modification date of the file after the copy. In case there was a modification, one really gets a lock and retries.

snapshots: One could use the database or file-system services to produce a consistent file snapshot (i.e. keep an old version of the file until the copy process is finished, but allow writers already to modify).

Any of these methods could be used to coordinate the access between local clients and the replication system in order to obtain an internally consistent file copy. This consistency level can be supported by a Grid middleware system but needs a mechanism for establishing locks to distributed files. Consequently, our proposed consistency service needs to support distributed file locking.

6.2 Consistent File Copy (Consistency Level 0)

At this consistency level, the data within a given file corresponds to a snapshot of the original file at some point in time. In the case of a file which is controlled by a database, the moment when the snapshot is taken may fall in the middle of one or more ongoing transactions. Again, we have the dirty read problem.

In this case, it is still unclear if a file copy in this intermediate state would be usable by a remote Grid user. The *complete* state of relational or object database transactions that are distributed over several data items consists of:

- the data in database files on disk
- the previous state of uncommitted changes as kept in journal or log files
- the state of locks as kept in the database server

Copying only part of the transactional state, namely only the database file, can therefore not assure consistency at the replica site. One can make the simplifying assumption that the complete state is contained *only* in the database file. Therefore, in principle only in this case can one hope to produce a consistent replica version which would be usable by a different client at another site.

Depending on the database implementation, there are again several mechanisms to obtain such a replica:

locks: one obtains a database read lock, depending on the database to exclude other writers.

snapshot: one instructs the database to maintain for the duration of the copy a transactionally consistent snapshot (e.g. using a MROW (multiple-reader-one-writer) read lock in an Objectivity based system). This would allow a concurrent writer to continue its work.

Not surprisingly, the requirement of replicating a consistent database state results in a similar situation as for simple files, but now any locks or snapshots have to be integrated with the database system instead of the file system.

6.3 Consistent Transactional Copy (Consistency Level 1)

Each replica has been produced at a time when no write transactions were active and can be used by other clients without internal consistency problems. However, if a Grid job requires more than just a single file, it may still experience inconsistency problems between these files. This time they could take place in the object model, e.g. when a data object in one file contains references to objects in another file that has been deleted, updated or relocated in between the two copy operations.

The following transaction sequence produces a consistent main store, but inconsistent replicas containing "dangling pointers".

- **starting point:** File A contains object a, which points to object b in file B. A job somewhere requires both files to be replicated to it.
- **t1:** file A gets copied first
- **t2:** some local transaction removes a and b, resulting in a new consistent state
- **t3:** file B gets copied

The result is the following. The main store is fine (a and b are consistently deleted). The replica is broken, since it contains object "a" pointing to non-existing object "b" (1) since the DBMS might actually reuse the object ID of the deleted object b, one might even end up with another complete unrelated object (probably an instance of different class now) (2) user code may still core dump or just compute wrong results.

One way to work around the problem of "dangling references" to objects in other files is to produce all replica files as part of a single database transaction. Also simpler approaches have been proposed, like removing cross-file references as part of the replication procedure. Since this approach may require a significant redesign of user applications to handle cross file references using other mechanisms, it may not generally be applicable.

6.4 Consistent Set of Transactional Copies (Consistency Level 2)

If the replicas have been produced as part of a single database transaction, the main consistency problem left is that replicated data might not be up to date, once the remote node starts working on it. Since replica and original are not part of a common database system, they are free to diverge. This in particular poses problems if it is required to merge the data changes from different sites to the same data. Note that we assume that at each site an independent data store is used that does not know about replicated sites. Therefore, replica and original are on different sites and in a different DBMS. In other words, a local DBMS only manages one instance of a physical file. A replica of this file is stored at a different site and managed by the local DBMS at this site.

6.5 Consistent Set of up-to-date Transactional Copies (Consistency Level 3)

This is basically what is called a "replicated federation" in Objectivity/DB where (1) a replica stays under the control of the database system and depending on the database implementation (2) read/write locks may have to be negotiated over the WAN. This often results in complex locking and recovery procedures, i.e. locks need to be removed in more than one replica location.

This is classical database replication as outlined in section 5.1. The DBMS manages all replicas and the access to data. In a Grid system, such a complex replication environment can only be attained if all data access operations use a common interface and do not allow non-Grid access like local fseek on files. This vision would mean that the Grid is a distributed database management system on its own but it may not be feasible for most of the Data Grid applications.

Read or write access to replicas is always atomic with a conventional database transaction. This is a very strict model and known as synchronous replication [11] which might be useful for some meta data but also may impose severe performance and usability constraints if applied to event data which has data volumes up to several Petabytes.

7 Merging Policies with Consistency

We now relate the possible policies to the consistency levels mentioned in the previous section and give a classification of the consistency levels.

For read-only data, none of the conventional inconsistencies illustrated in the previous section can occur. All the files will always have a consistent state. No transactions are required for update propagation to other replicas and maximum internal consistency can be achieved. However, there is still the issue of meta-data consistency on file creation. In Section 5 we have assumed that data already exists and have discussed the update problem. However, we need to discuss also the data creation step in order to catch all possible inconsistent states in a Data Grid. Therefore, we categorise the following two consistency problems and assign the responsibility to Grid services.

- data creation
- updates of existing data

As regards replication of read-only files, the only consistency requirement is that files are created and have unique names. This uniqueness criterion has to be checked and guaranteed by the replica catalogue service and is a low level consistency problem. If objects within a file need to be exposed to the Grid, these objects need to be uniquely identified too. Local uniqueness has to be guaranteed by the DBMS.

Synchronous and asynchronous replication protocols presented in 5.1 are a clear task for a higher-level consistency service on top of the replica manager and the replica catalogue service. Whereas synchronous replication approaches guarantee a strict consistency level 3, in many cases it is enough if sites are updated asynchronously. Asynchronous replication corresponds to consistency level 2 if multiple files contain certain associations or relevant mutual information. In this case a collection of files is considered to be in a consistent state (see Section 5.2). If files are independent of each other, asynchronous replication can also relax the consistency level a bit more and just provide level 1 consistency.

In general, level 1 consistency is the minimal consistency provided by a database management system that uses transactions.

Consequently, a replication (copy) operation always has to be regarded as a database read transaction that can only be executed successfully if a read lock is gained. A Grid consistency service thus has the following steps for consistent file replication:

1. gain read lock on all replicas of the same file in the Grid (use the replica catalogue for finding out the file locations).
2. transfer data securely
3. insert the file information into the replica catalogue
4. release read lock

A file update operation is more complex and needs write locks on the file. Several mechanisms like a quorum have been proposed [22] to reduce the amount of replicas to be available in order to have a successful write lock on a set of replicas.

8 Conclusion

We believe that having different consistency levels for replication is useful and possible if knowledge about data and use cases is available. We have described several different consistency models (with their consistency guarantees and their impact on the replication implementation) which we could offer to the Grid user. However, we do not impose any particular consistency constraints but leave it to the user to decide which consistency model is adequate for a particular application. Consistency guarantees expected for meta-data should be included. Level -1 does not seem to be applicable for many applications, and level 0 is probably excluded at least for database controlled files.

It is useful to allow an application to specify (possibly back-end specific) which level of consistency it expects for the requested file set. Such high-level replication services satisfy the need of a particular user community and make use of lower Grid replication services like replica catalogue management and optimised and secure file transfer.

In short, achieving even limited consistency produces considerable additional complexity if the replication system works directly on a data store containing multiple interrelated writable files. In the near future we will continue our analysis of feasible consistency models for writable data and extend it to other “data exchange options” than just database files. For read-only data, transferring database files is of course simple and very effective.

Acknowledgement

We want to thank Fons Rademakers and Brian Tierney for useful discussions on the paper.

References

[1] Divyakant Agrawal, Amr El Abbadi, R. Steinke. Epidemic Algorithms in Replicated Databases. *16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, May 12-14, 1997.

[2] Bill Allcock, Ann Chevernak, Ian Foster, Carl Kesselman, Chuck Salisbury, Steve Tuecke. The Data Grid: Towards an Architecture for Distributed Management and Analysis of Large Scientific Data Sets. to be published in the *Journal of Network and Computer Applications*.

[3] Bill Allcock, Joe Bester, John Bresnahan, Ann Chervenak, Ian Foster, Carl Kesselman, Samuel Meder, Veronika Nefedova, Darcy Quesnel, Steve Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing, *18th IEEE Symposium on Mass Storage Systems and 9th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, April 17-20, 2001.

[4] Yuri Breitbart, Henry Korth. Replication and Consistency: Being Lazy Helps Sometimes, *16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, May 12-14, 1997.

[5] Dirk Düllmann. Workshop presentation on issues with multiple synchronised federations, http://wwwinfo.cern.ch/asd/rd45/workshops/july99/ Multi-FD-Issues/sld001.htm

[6] European DataGrid Project, http://www.eu-datagrid.org

[7] Roy Fielding, James Gettys, Jeffrey Mogul, Henrik Nielson, Larry Masinter, Paul Leach, Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, June 1999.

[8] Ian Foster, Carl Kesselman. A Data Grid Reference Architecture. In preparation. 2001.

[9] Globus Project. GridFTP - Universal Data Transfer for the Grid. White Paper. September 5, 2000.

[10] Jim Gray, Andreas Reuter. Transaction Processing: Concepts and Techniques. The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor, 1993.

[11] Jim Gray, Pat Helland, Patrick O'Neil, Dennis Shasha. The Dangers of Replication and a Solution. *ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.*

[12] GriPhyN Project, http://www.griphyn.org

[13] Koen Holtman, Prototyping of CMS Storage Management, Ph.D. thesis, CERN, Geneva, Switzerland, 2000.

[14] Wolfgang Hoschek, Javier Jean-Martinez, Asad Samar, Heinz Stockinger, Kurt Stockinger. Data Management in an International Data Grid Project. *1st IEEE/ACM International Workshop on Grid Computing (Grid'2000)*. Bangalore, India, Dec 17-20, 2000.

[15] Matthias Nicola, Matthias Jarke. Increasing the Expressiveness of Analytical Performance Models for Replicated Databases, *International Conference on Database Theory (ICDT'99)*, Jerusalem, January 1999.

[16] Objectivity Inc., http://www.objectivity.com

[17] Oracle: http://www.oracle.com

[18] Jon Postel, Joyce Reynolds. RFC 959: File Transfer Protocol (FTP), October 1985.

[19] PPDG (Paricle Physics Data Grid), http://www.ppdg.net

[20] RD45 A Persistent Object Object Manager for HEP, http://wwwinfo.cern.ch/asd/rd45

[21] Asad Samar, Heinz Stockinger. Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication, *IASTED International Conference on Applied Informatics (AI2001)*, Innsbruck, Austria, February 19-22, 2001.

[22] Heinz Stockinger, Data Replication in Distributed Database Systems, CMS Note 1999/046, July 1999.

[23] Heinz Stockinger. Distributed Database Management Systems and the Data Grid. *18th IEEE Symposium on Mass Storage Systems and 9th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, April 17-20, 2001.

[24] The Object Data Standard: ODMG 3.0 R.G.G Cartell and Douglas K. Barry (editors), Morgan Kaufmann

[25] The ROOT System, http://root.cern.ch/